Computing Smooth Feedback Plans Over Cylindrical Algebraic Decompositions

Stephen R. Lindemann and Steven M. LaValle
Department of Computer Science
University of Illinois
Urbana, IL 61801 USA
{slindema, lavalle}@uiuc.edu

***Abstract*— In this paper, we construct smooth feedback plans over cylindrical algebraic decompositions. Given a cylindrical algebraic decomposition on $\mathbb{R}^n$, a goal state x_g, and a connectivity graph of cells reachable from the goal cell, we construct a vector field that is smooth everywhere except on a set of measure zero and the integral curves of which are smooth (i.e., C^∞) and arrive at a neighborhood of the goal state in finite time. We call a vector field with these properties a smooth feedback plan. The smoothness of the integral curves guarantees that they can be followed by a system with finite acceleration inputs: $\ddot{x} = u$. We accomplish this by defining vector fields for each cylindrical cell and face and smoothly interpolating between them. Schwartz and Sharir showed that cylindrical algebraic decompositions can be used to solve the generalized piano movers' problem, in which multiple (possibly linked) robots described as semi-algebraic sets must travel from their initial to goal configurations without intersecting each other or a set of semi-algebraic obstacles. Since we build a vector field over the decomposition, this implies that we can obtain smooth feedback plans for the generalized piano movers' problem.**

I. Introduction

Feedback motion planning is a fundamental problem in control and robotics. If the state space is a smooth manifold $X \subseteq \mathbb{R}^n$, and the system satisfies the state transition $\dot{x} = f(x, u)$, a *feedback strategy* (also called a *control law*) is a map $\pi : X \rightarrow \mathcal{U}$, in which $\mathcal{U}$ is the input space. A feedback strategy can also be seen as a vector field on X, since the choice of u at any point $x \in X$ determines the tangent vector of the system trajectory at that point. For a feedback strategy to be useful, the behavior of the system under the strategy must have some desirable properties. For example, stability is an important consideration; the control law should prevent the system from being unbounded as time goes to infinity. Another important property is convergence to a given goal point or region. For this to be satisfied, the system should be guaranteed to converge to the goal region under application of the feedback control. In this paper, we consider feedback motion planning on the cells of a cylindrical algebraic decomposition of a bounded subset of $\mathbb{R}^n$. The system we consider is of the form $\ddot{x} = u$. If our cylindrical algebraic decomposition arises from a generalized piano movers' problem as in Schwartz and Sharir [1], with some cells of the decomposition removed because of the configuration space obstacles, then the feedback plan (vector field) we construct is guaranteed to take any initial state to the goal state while avoiding the obstacle cells, and to do so smoothly. Hence, our method computes a smooth feedback plan for the generalized piano movers' problem.

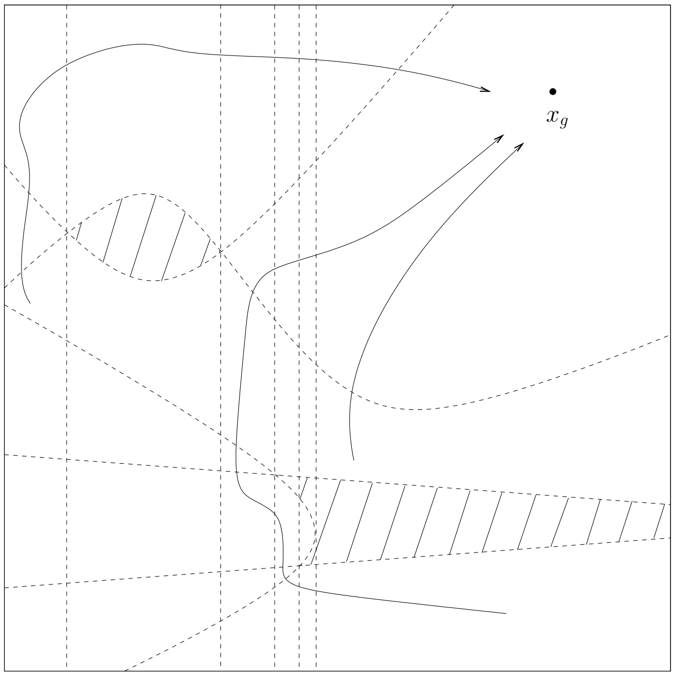

Fig. 1. The resulting cylindrical algebraic decomposition and several flows of a smooth feedback plan on the cell decomposition.

Traditional feedback control is well studied [2], but cannot be applied in many cases due to nonconvex obstacles in the environment. This is difficult enough when we are considering nonconvex obstacles in the plane; it is far more challenging when the obstacles are the semi-algebraic sets in a high-dimensional configuration space corresponding to the generalized piano movers' problem. In the algorithmic motion planning community, the solution to this has been to compute open loop trajectories linking initial and goal configurations but ignoring feedback considerations. This work is very important because it provides a way to compute trajectories for very complex, high-dimensional problems; however, it is important to think not only in terms not only of open loop trajectories, but also about feedback control.

Some have tried to make feedback more central through the construction of potential fields that have no local minima other than the goal state [3], [4]. If such a potential field can be found, the gradient of the field can be used as the velocity command for the robot. However, there are a number of difficulties associated with computing such potential fields. We will bypass these difficulties by directly constructing a vector field with the desired properties, rather than constructing a real valued function and using the gradient as the vector field. In our case, we take a cylindrical algebraic decomposition of the configuration space (which is known to be able to solve the generalized piano movers' problem [1]) and construct vector fields for each of the cells in the decomposition. We do this by inductively defining vector fields on cells of dimension one and then iteratively lifting them into more dimensions, in the same way as the decomposition itself is constructed. Different locally defined vector fields are smoothly combined using bump functions. The result is a globally defined vector field the integral curves of which are smooth and converge to a goal state. An illustration of a vector field produced by our algorithm is given in Figure 1. Our vector fields can be used directly for kinematic systems, or they can be used to develop dynamic control policies. For example, if the computed vector field is $V(x)$, a control policy

$$u = K(V(x) - \dot{x}) + \dot{V}(x)$$

for some feedback gain K can be used [5]. Under certain conditions, it can be shown that the system will converge to the integral curves of the constructed vector field [5], [6].

In the following section, we will review related work, focusing on how the feedback motion problem has been addressed within the robotics community and describing in detail the method of upon which ours is based. We will then briefly describe cylindrical algebraic decomposition, and our algorithm in detail. We will demonstrate that the integral curves of our vector field converge to the goal state.

II. Feedback Motion Planning

A. Background

The problem of finding a global motion plan in complex environments is difficult. Motion planning problems in robotics typically involve non-convex constraints resulting from obstacles in the environment. This presents a significant problem for traditional feedback control methods. One solution might be to use state space sampling along with dynamic programming to achieve not only feedback, but approximately optimal trajectories [7]–[9]. This may be feasible for low-dimensional spaces, but both the time- and space-complexity is exponential in the dimension of the state space, assuming that the sampling resolution remains fixed. The difficulty of feedback control for these problems motivates the development of open loop motion planning algorithms, which can at least find feasible paths through obstacle-cluttered environments. Such algorithms have been extensively studied [10], [11]. Many motion planning algorithms have been developed for kinematic systems; several, such as RRTs [12] and PDST-EXPLORE [13] are specifically designed for systems with dynamics. Kinematic motion planning algorithms find paths which need post-processing (e.g., time-scaling [14], [15], steering [16], [17], or other transformations [18], [19]) to be transformed into trajectories for dynamical systems. In contrast, RRTs and similar planners find such trajectories directly. In either case, an open loop trajectory for the system is found. This trajectory can then be tracked using feedback.

This approach has several disadvantages, however. First, paths generated by motion planning algorithms often appear to be of poor quality, having unnecessary turns and bends in them. This may result in them being difficult to follow for a dynamical system. Second, this approach does not produce a global feedback plan, but only a local feedback plan in a neighborhood of the nominal trajectory. It would be better to solve the feedback problem once for the entire space.

Another approach, made plausible through tremendous advances in computational power, is to use motion planning algorithms themselves as the feedback mechanism. In such a model, any time the system deviated from the prescribed trajectory, the trajectory would be re-planned (probably from scratch) based on the new state of the system. This approach is extremely problematic as well. First, it has a very high computational cost, and may not be suitable for real-time applications. Second, this approach is not even guaranteed to bring the system to the goal state, although in practice it might be expected to.

These approaches, which add feedback almost as an afterthought to open loop trajectories, have significant problems, as we have seen. Consequently, there have been some attempts within the robotics community to incorporate feedback more directly. For example, the sampling-based neighborhood graph (SNG) covers the free space with balls, each of which is equipped with a local navigation function which is guaranteed to convey the robot into a ball nearer to the goal state. Other approaches to feedback motion planning in the presence of obstacles are often based on potential fields. Khatib [3] developed a method which utilized a potential field over the operational space to guide a manipulator or mobile robot to the goal. His approach suffers from local minima, however, as do many potential field methods. Waydo and Murray give a stream function method for navigation in two-dimensional environments [20]. A highly influential potential field method is that of Rimon and Koditschek [4], who show how to develop *navigation functions* (potential functions with a unique minimum at the goal and meeting certain other criteria) using potential functions in a generalized sphere world. Rimon and Koditschek have presented the most general feedback planning technique up to now; their method applies to any problem whose configuration space is topologically equivalent to a generalized sphere world. Our method is more general in that it applies to any configuration space with a well-defined cylindrical algebraic decomposition.

Finally, work by Conner *et al.* [6] and Lindemann and LaValle [21] compute feedback plans over cell decompositions, as do we. Conner *et al.* consider an cell complex environment in d-dimensional Euclidean space. They then impose a potential field over each individual cell, taking as the field the pullback of a potential function on a disk, which has a closed form solution. They require that the gradients of the potential fields be perpendicular to the cell boundaries, so that adjoining potential fields can be easily pieced together. Putting the individual "component control policies" together guarantees that the global control policy brings the robot to the goal. In addition to specifying a control policy for kinematic systems, they develop control policies for systems with dynamical constraints. Similarly, Lindemann and LaValle take a cell complex environment and define vector fields over the individual cells, which can also be see as component control policies. Since this work is the primary inspiration for the current work, we describe it in detail below. Both [6] and [21] can be seen in the context of the sequential composition of funnels approach [22], in which a collection of controllers is developed, each of which converges to a goal set which is either the actual goal state or in the domain of another controller. Following a sequence of these controllers will cause the system to arrive at the goal state. This idea was further developed in [5], [23].

B. Smooth Feedback Plans on Convex Polytopes

Lindemann and LaValle introduced the method upon which this work is based [21]. We will describe their work in depth, especially the parts in which our method parallels theirs. They address the problem of smooth feedback motion planning for a point robot whose environment is a d-dimensional cell complex, each cell being a bounded d-dimensional convex polytope. Such a cell complex might be generated from a convex decomposition of a d-dimensional polygonal environment. There is a goal state x_g, and consequently a goal cell C_g containing x_g. They use the connectivity of the convex cells to construct a graph and use a graph search algorithm (such as Dijkstra's algorithm or breadth first search) to determine a path from each cell to C_g. Then, each cell other than C_g has a "successor" cell which is the next cell on the path to the goal cell.

Once the cell graph has been computed, they construct a vector field on each cell which has the following properties:

1) The vector field is smooth except for a set of measure zero and all integral curves of the vector field are smooth.
2) All integral curves leave the cell via the exit face, entering the designated successor of that cell.
3) Smoothness of the integral curves is preserved when cell boundaries are crossed.

These properties guarantee that the vector field can be used for smooth feedback motion planning for the system $\ddot{x} = u$.

An important element of the method is the use of the fact that smooth feedback planss can be constructed using two types of simple vector fields, one per d-dimensional cell and one per $(d-1)$-dimensional face, blended together using bump functions. Since bump functions smoothly interpolate between two functions (more generally, partitions of unity are capable of smoothly blending arbitrarily many functions together), the system can transition from following one component vector field to another without any loss of smoothness. Bump functions are defined as follows:

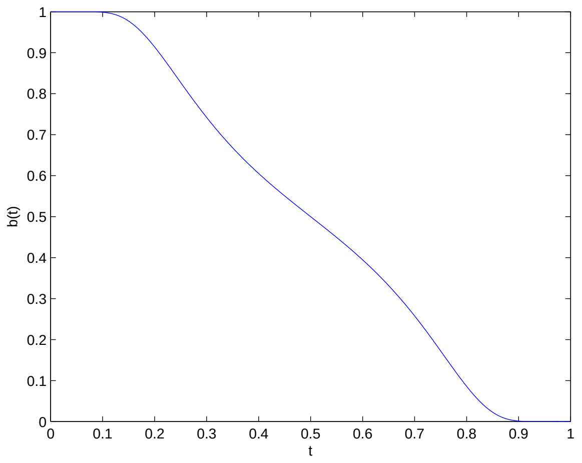

Fig. 2. A bump function. If we let $\lambda(t) = (1/t)e^{-1/t}$, then $b(t) = 1 - \lambda(t)/(\lambda(t) + \lambda(1-t))$.

Definition 1 *Let X be a smooth manifold, and let K be a closed set and U an open set, $K \subset U \subseteq X$. A bump function over U is a smooth, real valued function $\rho : X \to [0,1]$ such that:*

1) *ρ has support contained in U.*
2) *$\rho(x) = 1$ for every $x \in K$.*

Bump functions are in general difficult to find, even though their existance is guaranteed. However, they are simple to compute on the real line. Figure 2 illustrates a bump function which smoothly transitions from 1 to 0 on the unit interval.

In order for the approach to work, one more piece must be put in place. Within each cell, the vector fields must be blended in such a way so that on each face, the resulting vector field is identically equal to the face vector field. In the interior of each cell, all of the face vector fields must be smoothly interpolated between. They use the cell vector field (which they call the *attractor* vector field) as an intermediary between the different face vector fields, interpolating between them. They do this using the interior generalized Voronoi diagram (GVD) of the cell. The GVD partitions the cell into regions corresponding to each face; in each region, bump functions are used to blend between the face vector field and the attractor vector field. On the GVD itself, the vector field is identically equal to the attractor vector field, guaranteeing that all face vector fields are smoothly blended together in the interior of the cell. In order to use the bump function to blend between the face vector field and the attractor vector field, they have to carefully construct the parameter of the bump function in such a way that it will be equal to one on any of the faces of the GVD and zero on the face of the cell. They use the product of all fractional distances to the faces of the GVD to do this. Formally, for

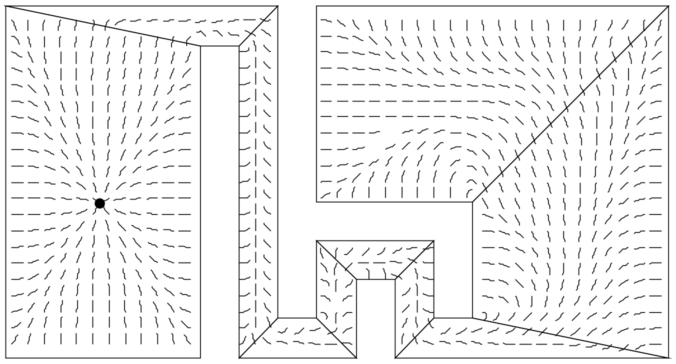

Fig. 3. A smooth feedback plan generated using the algorithm of [21] in a two-dimensional environment.

any point p it is defined as

$$t(p) = 1 - \prod_{i=1}^{n} \frac{d(p, f_i)}{d(p, f_i) + d(p, f_x)}, \tag{1}$$

in which $\{f_i\}_1^n$ are the faces of the GVD and $d(p, f_i)$ is the distance from p to face f_i. Note the similarity to the analytic switch used in navigation function and potential field methods. This function is smooth (except at the vertices of the cell), and has the desired property of being identically equal to one on the exit face and zero on all other boundaries.

Since the bump function smoothly blends the face and attractor vector fields together, a vector field is obtained which is smooth over the entire cell. With small modifications, the above approach can be used in the goal cell as well. They show that piecing the individual cells together results in a vector field which is smooth over the entire free space. They also normalize the vector field at every point, so the global vector field $V(p)$ is defined as $V(p) = \mathrm{norm}(b(p)V_f(p) + (1 - b(p))V_a(p))$, in which V_f is the face vector field for that point, V_a the attractor vector field, b the bump function, and norm is the normalization function. An example of the trajectories produced by the method is given in Figure 3.

III. Cylindrical Algebraic Decomposition

Cylindrical algebraic decomposition was used to solve the piano movers' problem first by Schwartz and Sharir [1]. They used the Collins decomposition [24] to partition the configuration space into free and obstacle cells and demonstrated how to find a path from a point in one free cell to a point in any other connected free cell. Cylindrical algebraic decomposition (abbreviated CAD) is extremely powerful; in fact, it is capable of solving the first-order theory of the reals [25]. Given this, it is not surprising that it can solve the generalized piano movers' problem as well.

The input to the CAD algorithm is set of polynomials. The polynomials are taken from the set $\mathbb{Q}[x_1, \ldots, x_n]$, the polynomials over the field of rational numbers $\mathbb{Q}$. The algorithm takes this set and computes a decomposition such that each cell is semi-algebraic and sign-invariant under the given polynomials. Each cell, then, is formed by the intersection and union of a finite number of these polynomials. There are two main phases to the CAD algorithm. First, the polynomials are iteratively projected downward, one dimension at a time, until a single dimension is reached. After reaching a single dimension, the real line is partitioned using the critical points of the projected polynomials. Then, each segment in the partition is lifted back up into $\mathbb{R}^2$, becoming cylinders which are partitioned based on the critical points of the now two-dimensional polynomials. This is repeated, each time lifting up and partitioning the resulting cylinders, until $\mathbb{R}^n$ is reached. At that point, a sign-invariant partition of $\mathbb{R}^n$ has been obtained. Figure 7 illustrates this. More details can be found in [11], [25], [26].

It is helpful to formally define the structure of a cell produced by the cylindrical algebraic decomposition algorithm. The definition is inductive:

Definition 2

1) *A cylindrical algebraic cell C_1 in dimension one is either an interval (a, b) or a point a.*
2) *A cell C_n in dimension n has one of the two forms: it is either the set of pairs $\{(x, y) : x \in C_{n-1}, f(x) < y < g(x)\}$ or the set of pairs $\{(x, y) : x \in C_{n-1}, y = f(x)\}$, in which f and g are the $(n-1)$-dimensional projections of the original polynomials.*

IV. Smooth Feedback Plans on Cylindrical Cells

To construct a smooth feedback plan on the entire cell decomposition, we construct smooth plans on each individual cell and then guarantee that smoothness is preserved across cell boundaries. We assume that the input to our algorithm is the entire cylindrical algebraic decomposition (i.e., the n-dimensional sign-invariant cells and their corresponding bounding polynomials), as well as a connectivity graph corresponding to the connectivity of the n-dimensional cells in the decomposition. If the goal cell is denoted as C_g, then we can search this graph to find a cell path from any other cell to the goal cell. Defining the successor of a cell C as the next cell on the path to C_g from C, we can construct a directed acyclic graph that describes the cell paths from every cell, each of which ends at the goal cell. For any cell C, the task is to construct a vector field such that all integral curves in C are smooth in C and exit into the successor cell of C, and to ensure that if any flows enter C from other cells, their smoothness is preserved (i.e., smoothness must be preserved along all integral curves).

In this section, we will describe our method for constructing the vector field and prove that the integral curves associated with it are smooth and converge to the goal state. We described in Section III how cylindrical algebraic decompositions can be generated by projecting the algebraic surfaces into progressively lower dimensions until only one dimension remains. At that point, the first cylinder cross-sections are given by the intervals; then, the surfaces are lifted up into successively higher dimensions, being divided into cylinders of free or occupied cells each time. Our algorithm proceeds analogously, beginning with the one-dimensional case, designing appropriate vector fields and then lifting them into higher dimensions.

Before continuing, we need to clarify the notation we will be using. We will discuss our algorithm in terms of an n-dimensional cell C_n, and its successor S_n. We also consider the cross sections of the two cells, which are denoted as C_{n-1} and S_{n-1}. The cross sections are taken perpendicular to the most recently lifted dimension (informally denoted the vertical dimension). The $(n-1)$-dimensional face between the two cells will be denoted F_{n-1} and its $(n-2)$-dimensional projection as F_{n-2}. Note that F_{n-2} is the intersection of the closures of C_{n-1} and S_{n-1}. Also, we have upper and lower bounding polynomials for C_n and S_n, which we will call g_C, f_C, g_S, and f_S, respectively. Note that we assume that C_n and S_n are in neighboring cylinders; if they are in the same cylinder, then the problem becomes very easy. We discuss this case later.

If general position assumptions are not made, then the connectivity of the cells of neighboring cylinders becomes very difficult to characterize; very little can be said about the behavior of the upper and lower bounding polynomials of the two cells. As in the seminal work of Schwartz and Sharir [1], we make the assumption that the polynomials are in general position. The connectivity of neighboring cells becomes much easier to analyze. Consider the cells C_{k-1} and S_{k-1}. Now, consider two cells C_k and S_k in the cylinders formed by lifting C_{k-1} and S_{k-1} into $\mathbb{R}^k$. If C_k and S_k are adjacent, then general position requires that they share either their upper or lower bounding polynomial, or both. If they do not share either bounding polynomial, they will not be adjacent. Furthermore, if only a single bounding polynomial is shared, then when C_k and S_k are lifted in higher dimensions, all adjacent cells will share both bounding polynomials. This means that two adjacent cells C_n and S_n in $\mathbb{R}^n$ share either all bounding polynomials in all dimensions, or all but one. This greatly simplifies the construction of the desired vector field. See Figure 4 for an illustration of adjacent cells not sharing a bounding polynomial.

One final set of definitions is necessary before proceeding. First, for any point $p = (x, y) \in C_n$, define the relative height function $h_r : C_n \to \mathbb{R}$ as $h_r(p) = (y - f_C(x))/g_C(x) - f_C(x))$. Next, define a projection function $\pi : C_n \to F_{n-1}$. For the sake of clarity, we will not give a rigorous definition for this projection, but let π take a point in C_n to the point in F_{n-1} which preserves the relative heights in all dimensions, except the dimension being projected out. This is a smooth mapping.

We will construct an n-dimensional vector field using two types of $(n-1)$-dimensional vector fields defined on C_{n-1}. The first will be one which flows out of the appropriate $(n-2)$-dimensional face F_{n-2}, and the second is a field which is inward pointing on the entire $(n-2)$-dimensional boundary of C_{n-1}. We will also use "vertical" vector fields; i.e., positive and negative axis aligned vector fields in the lifted dimension. We will begin by constructing vector fields in one dimesion, and then proceed to the inductive step to n dimensions. As

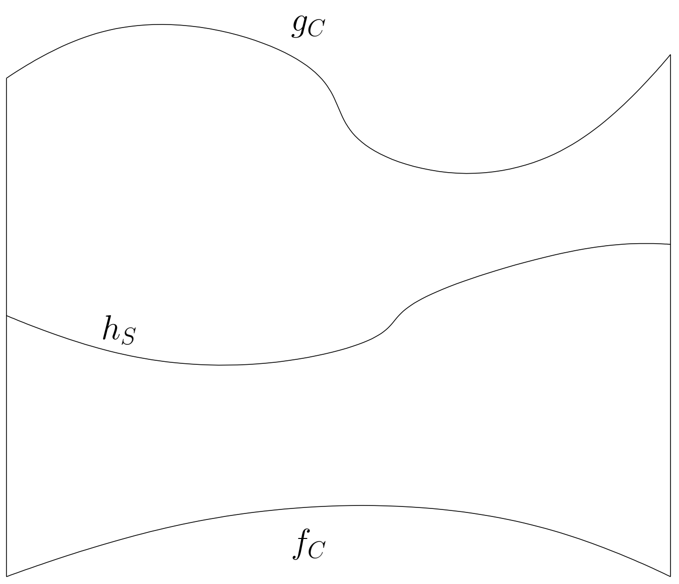

Fig. 4. A face separating two cells from adjacent cylinders. The functions f_C and g_C are the bounding polynomials for one cell, and h_S is either f_S or g_S, depending on whether the successor S is the upper cell or the bottom cell.

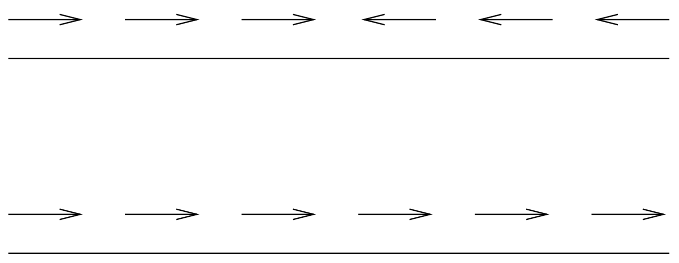

Fig. 5. The two types of vector fields on an interval. Above, an inward pointing field needing smoothing using a blending vector field; below, a smooth outward pointing vector field.

in the previous work described in Section II, these vector fields will utilize face vector fields and blending vector fields (referred to as attractor vector fields in [21]). We will also need to define a pseudo-distance function which we will use in construction the parameter of the bump function. For an n-dimensional point p, we will use the notation $d_n(p)$ for the distance function.

In one dimension, the construction is obvious, and is illustrated in Figure 5. For the inward pointing field, each endpoint has an associated inward pointing vector field; these two vector fields are interpolated using a blending vector field, which we leave unspecified for the moment. The distance function is the obvious Euclidean distance. If the cell C_1 (i.e., line segment) has endpoints e_1 and e_2, we can formally define $d_1(p) = \min(d(p, e_1), d(p, e_2))$, in which d is the standard Euclidean metric. For the outward pointing field, we do not need a blending field; the vector field always points in the same direction.

Now we proceed to the inductive step. Assuming we have a smooth vector field for the $(n-1)$-dimensional cell C_{n-1}, we need to construct an appropriate vector field on the lifted n-dimensional cell. Recall the structure of the cell as given in Section III; each point p in the cell C_n is a pair (x, y), in which x is in the $(n-1)$-dimensional cell C_{n-1} and $f_C(x) < y < g_C(x)$. To smoothly interpolate between the vertical vector field and an $(n-1)$-dimensional vector field, we need a distance function on C_{n-1} that is smooth. Assume we have a smooth distance function for the cell C_{n-2}. Then we can define

$$t_{n-1}(p) = 1 - \frac{d_{new}(p)}{d_{new}(p) + d_{n-2(p_{n-2})}},$$

in which $d_{new} = \min(|g(x) - y|, |f(x) - y|)$, g and f the upper and lower bounding polynomials in C_{n-1}, and p_{n-2} the projection of the point p into C_{n-2}. Then define

$$d_{n-1}(p) = b(t_{n-1}(p))d_{n-2}(p_{n-2}) + (1 - b(t_{n-1}(p))d_{new}(p),$$

in which b is the bump function seen in Figure 2. This function has the desired smoothness properties and can be used in the parameter of the blending bump function in the cell C_n.

Our central approach is to decouple the n-dimensional problem into two subproblems: first, a one-dimensional problem in the new lifted dimension; and second, an $(n-1)$-dimensional problem which has already been solved by the lower dimensional constructions. We make the following definition:

Definition 3 *The exit window of cell C_n exiting via face F_{n-1} is the set of points:*

1) $\{x \in C_n : h_r(g_S(\tau(x))) < h_r(x)\}$, *if* $f_S(y) = f_C(y)$ *and* $g_S(y) < g_C(y)$ *for all* $y \in F_{n-1}$.
2) $\{x \in C_n : h_r(x) < h_r(g_S(\tau(x)))\}$, *if* $g_S(y) = g_C(y)$ *and* $f_S(y) > f_C(y)$ *for all* $y \in F_{n-1}$.
3) $\{x \in C_n\}$, *otherwise.*

In the definition, the first case corresponds to S_n having the same lower bounding polynomial as C_n but an upper bounding polynomial which is lower, and the second case corresponds to S_n having the same upper bounding polynomial but a lower bounding polynomial which is greater. Recall that at most one bounding polynomial can be different. In the third case, the entire cell C_n is the exit window. Intuitively, the exit window consists of all points who have a relative height between those of the upper and lower bounding polynomials through which they must pass.

The upper and lower boundaries of this exit window are obvious, and these boundaries can be used to logically partition the cell C_n into at most two slices: denote the exit window as C_{exit} and the other as C_{hold}, if it exists. In order to define the vector field on each of these cells, we need to define vector fields for the upper and lower faces, the cross-section, and the blending vector field. In C_{hold}, the cross-section vector field must be inward pointing on the entire cross-section boundary; in the inductive step, we assumed that we had one of these available (call it V_{in}). Since we know that for C_{exit}, following the lower dimensional exit vector field will lead to a proper cell

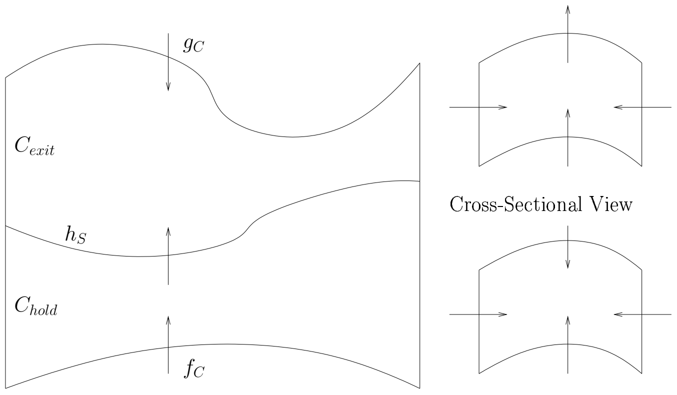

Fig. 6. The component vector fields for a three dimensional cell. The vertical face separating C and S is on the left, and we let $h_S = f_S$. For the lower part of C, the cross-section vector field is inward pointing; on the upper, it exits through the appropriate face.

exit, we define the cross-section vector field in that case to be V_{exit}. The vector fields for the upper and lower boundaries of the cells are simple, being either upward pointing or downward pointing, as appropriate. Finally, the blending vector field is chosen to make sure that the flows continue to the desired exit region. Within each cell, blend the component vector fields together using bump functions, the parameter of the bump functions being constructed in the same as in [21], which was described in Section II. It is fairly obvious that the upper and lower cells prevent the integral curves from leaving the cell except to enter C_{exit}, and that all flows in C_{exit} leave through the exit face as desired. The orthogonality of the vertical vector fields and the $(n-1)$-dimensional vector fields makes this relatively easy to show, although we will not give all the details here.

At this point, we have done most of the hard work to prove our inductive hypothesis. We have constructed a vector field using the vector field for C_{n-1} and the new vertical vector fields, which will guarantee that all flows in C_n leave from the appropriate exit face F_{n-1}. We have also recursively defined the pseudo-distance function used in the parameter of the bump function. Only two things remain. First, we need to be able to have vector field that leaves out of the top or bottom of the cell into another cell in the same cylinder. This is trivial; use upward pointing vector fields on both the top and bottom faces and an inward pointing field on C_{n-1}. The rest of the construction is identical. Finally, we also need to be able to constructan inward pointing vector field on C_n. This too is trivial; use V_{down} on the top face, V_{up} on the bottom face, and the inward pointing field on C_{n-1}. Note that the integral curves crossing cell boundaries will remain smooth; this is because the vector field is always axis aligned on the cell boundaries, so adjacent vector fields match perfectly.

All that remains is to show how to create a vector field for the goal cell, which instead of exiting through an exit face, should converge to a point. This is fairly simple, being just a variation on the inward pointing cell just described. We do not describe it in full, but it is simple to construct a vector field in one dimension that converges to a point on the interval. This can be extended inductively by causing the vertical vector fields to converge to the desired coordinate together with the lower dimensional vector field.

We have shown how to construct a vector field over the entire cylindrical algebraic decomposition so that all the integral curves are smooth and converge to the goal state. This is a smooth feedback plan. Consequently, a trajectory beginning anywhere in the configuration space will be smooth and will reach an arbitrary neighborhood of the goal state in finite time. We summarize the result:

Theorem 1 *For a given cylindrical algebraic decomposition, goal state x_g, and and a connectivity graph of cells reachable from the goal cell, we have constructed a vector field V such that:*

1) *it is smooth everywhere except on a set of measure zero;*
2) *the integral curves are smooth; and*
3) *the integral curves arrive at a neighborhood of the goal state in finite time.*

Therefore, we have obtained a smooth feedback plan on the decomposition.

One remaining question is that of the complexity of the algorithm. Our method is very efficient with respect to the input. Specifically, if the input is taken to be the complete cylindrical algebraic decomposition, including the cells of every dimension from 1 to n as well as a connectivity graph, then our method requires precomputation time $O(n)$ in the size of the input (it needs only to precompute the graph for the cell path, which can be done using breadth-first search in linear time). A trajectory can be initialized using naive point location in linear time; this bound can be improved in certain cases. The vector field can be computed at any given point in a known cell in time linear in the complexity of the cell itself. This justifies our earlier claim that we essentially obtain "feedback for free" once the cylindrical algebraic decomposition has been computed. However, the number of cells in the decomposition can be doubly exponential in the dimension of the configuration space, so these results do not have practical significance.

V. Conclusions

In conclusion, we have introduced an algorithm for constructing a vector field on the cells of a cylindrical algebraic decomposition. Since CAD algorithms solve very general motion planning problems (virtually any problem with a semi-algebraic robot and semi-algebraic obstacles, in fact) [1], [11], this implies that we can provide smooth feedback plans for these problems as well. To the authors' knowledge, this is the first construction of smooth feedback plans with this level of generality. It is also of interest that vector fields with smooth integral curves can be constructed with relatively little trouble and in an intuitive way.

Due to the complexity of cylindrical algebraic decompositions, it is highly unlikely that this method will be implemented and used in its full generality. However, it has more than purely theoretical interest. It may be reasonable to apply to feedback planning for the rod [27] or for polygonal robots translating and rotating in the plane [28]. Additionally, these ideas can be applied to other specialized cell decompositions, or used in conjunction with a precomputed path to provide feedback in a neighborhood of the path. In the future, we plan to explore these avenues of research.

References

[1] J. T. Schwartz and M. Sharir, "On the piano movers' problem: II. General techniques for computing topological properties of algebraic manifolds," *Communications on Pure and Applied Mathematics*, vol. 36, pp. 345–398, 1983.

[2] C.-T. Chen, *Linear System Theory and Design*. New York, NY: Holt, Rinehart, and Winston, 1984.

[3] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *Int. J. Robot. Res.*, vol. 5, no. 1, pp. 90–98, 1986.

[4] E. Rimon and D. E. Koditschek, "Exact robot navigation using artificial potential fields," *IEEE Trans. Robot. & Autom.*, vol. 8, no. 5, pp. 501–518, Oct. 1992.

[5] A. A. Rizzi, "Hybrid control as a method for robot motion programming," in *IEEE Int. Conf. Robot. & Autom.*, 1998, pp. 832–837.

[6] D. C. Conner, A. A. Rizzi, and H. Choset, "Composition of local potential functions for global robot control and navigation," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, 2003, pp. 3546–3551.

[7] D. Bertsekas, *Dynamic Programming and Optimal Control: Volume I*. Belmont, MA, USA: Athena Scientific, 2000.

[8] S. M. LaValle and P. Konkimalla, "Algorithms for computing numerical optimal feedback motion strategies," *International Journal of Robotics Research*, vol. 20, no. 9, pp. 729–752, Sept. 2001.

[9] J. N. Tsitsiklis, "Efficient algorithms for globally optimal trajectories," *IEEE Trans. Autom. Control*, vol. 40, no. 9, pp. 1528–1538, Sept. 1995.

[10] J.-C. Latombe, *Robot Motion Planning*. Boston, MA: Kluwer Academic Publishers, 1991.

[11] S. M. LaValle, *Planning Algorithms*. Cambridge University Press (also available at http://msl.cs.uiuc.edu/planning/), to be published in 2006.

[12] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," in *Algorithmic and Computational Robotics: New Directions*, B. R. Donald, K. M. Lynch, and D. Rus, Eds. Wellesley, MA: A K Peters, 2001, pp. 293–308.

[13] A. M. Ladd and L. E. Kavraki, "Fast exploration for robots with dynamics," in *Proc. Workshop on Algorithmic Foundation of Robotics*, 2004, pp. 313–328.

[14] J. E. Bobrow, S. Dubowsky, and J. S. Gibson, "Time-optimal control of robotic manipulators along specified paths," *Int. J. Robot. Res.*, vol. 4, no. 3, pp. 3–17, 1985.

[15] K. G. Shin and N. D. McKay, "Minimum-time control of robot manipulators with geometric path constraints," *IEEE Trans. Autom. Control*, vol. 30, no. 6, pp. 531–541, 1985.

[16] F. Lamiraux and J.-P. Laumond, "Flatness and small-time controllability of multibody mobile robots: Applications to motion planning," *IEEE Transactions on Automatic Control*, vol. 45, no. 10, pp. 1878–1881, Apr. 2000.

[17] R. M. Murray and S. Sastry, "Nonholonomic motion planning: Steering using sinusoids," *Trans. Automatic Control*, vol. 38, no. 5, pp. 700–716, 1993.

[18] J. P. Laumond, S. Sekhavat, and F. Lamiraux, "Guidelines in nonholonomic motion planning for mobile robots," in *Robot Motion Plannning and Control*, J.-P. Laumond, Ed. Berlin: Springer-Verlag, 1998, pp. 1–53.

[19] S. Sekhavat, P. Svestka, J.-P. Laumond, and M. H. Overmars, "Multilevel path planning for nonholonomic robots using semiholonomic subsystems," *Int. J. Robot. Res.*, vol. 17, pp. 840–857, 1998.

[20] S. Waydo and R. M. Murray, "Vehicle motion planning using stream functions," in *IEEE Int. Conf. Robot. & Autom.*, 2003, pp. 2484–2491.

[21] S. R. Lindemann and S. M. LaValle, "Smoothly blending vector fields for global robot navigation," in *Proc. IEEE Conference on Decision and Control*, 2005, pp. 3553–3559.

[22] T. Lozano-Pérez, M. T. Mason, and R. H. Taylor, "Automatic systhesis of fine-motion strategies for robots," *Int. J. Robot. Res.*, vol. 3, no. 1, pp. 3–24, 1984.

[23] R. R. Burridge, A. A. Rizzi, and D. E. Koditschek, "Sequential composition of dynamically dexterous robot behaviors," *Int. J. Robot. Res.*, vol. 18, no. 6, pp. 534–555, 1999.

[24] G. E. Collins, *Lecture Notes in Computer Science*. Berlin: Springer-Verlag, 1975.

[25] B. Mishra, "Computational real algebraic geometry," in *Discrete and Computational Geometry*, J. E. Goodman and J. O'Rourke, Eds. New York: CRC Press, 1997, pp. 537–556.

[26] S. Basu, R. Pollack, and M.-F. Roy, *Algorithms in Real Algebraic Geometry*. Berlin: Springer-Verlag, 2003.

[27] J. Bañon, "Implementation and extension of the ladder algorithm," in *IEEE Int. Conf. Robot. & Autom.*, 1990, pp. 1548–1553.

[28] F. Avnaim, J. D. Boissonat, and B. Faverjon, "A practical exact planning algorithm for polygonal objects amidst polygonal obstacles," in *IEEE Int. Conf. Robot. & Autom.*, 1988, pp. 1656–1660.

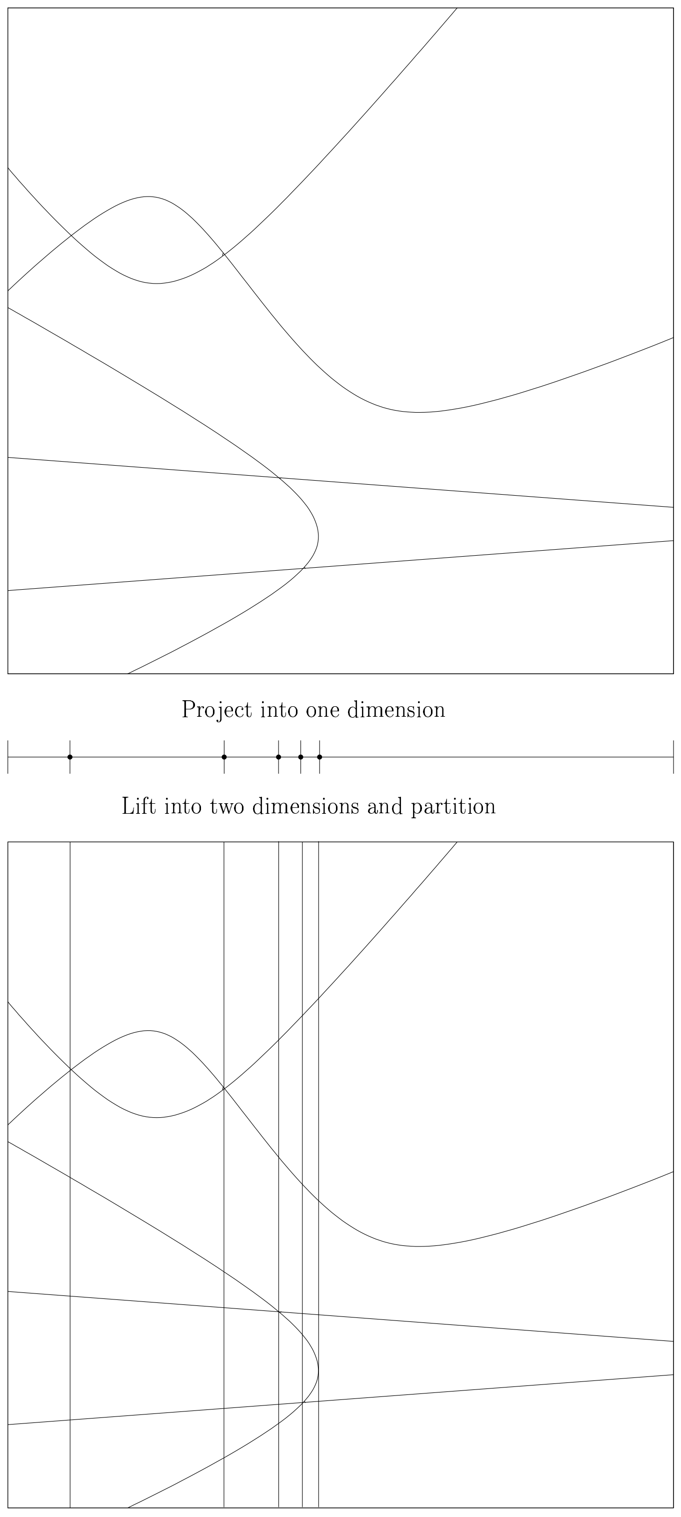
Project into one dimension
Lift into two dimensions and partition

Fig. 7. The polynomials corresponding to the obstacles from Figure 1, and the steps of the CAD algorithm.